Player-Missile Graphics

Objectives of Module

In this module you will learn how to create your own players (moveable objects on the screen) and you will learn to animate them. You will also explore how to change the size, shape, and color of your players.

Overview (subtopics)

1. What is Player-Missile Graphics?
   - Purpose of Player-Missile Graphics.
   - What is a player? A missile?
   - Features of players and missiles.

2. Player-Missile RAM.
   - What is P-M RAM?
   - Designing and moving players.

3. Vertical Motion of Players.

4. Setting Up Player-Missile Graphics.
   - How to turn on P-M graphics.
   - Changing resolution, width, and color.

5. Summary and Challenges.
   - Setting priorities.
   - Collision detection.

Prerequisite Understanding Necessary

1. You must know Atari BASIC.

Materials Needed

1. BASIC cartridge.

2. Advanced Topics Diskette.

What is Player-Missile Graphics?

The Atari computer has a very useful feature built into the hardware called Player-Missile graphics. In this section you will learn what players and missiles are, why they are useful, and some of their special properties.

Many of the interesting things one can do with a computer involve moving objects on the screen. This is especially true of many games that are played on the computer. One of the most straight-forward techniques for animating an object on the screen is to draw it, erase it, and then redraw it in a slightly shifted position. If you continue this process, the object appears to move across the screen.

Unfortunately it is rather cumbersome and slow to continually draw and erase graphics on the screen. To avoid this, the Atari computer was designed with a feature for moving objects on the screen more quickly and easily. There are some limitations to the size and number of objects that you can animate, but the limitations are usually worthwhile because of how much easier it is to make things move. Worksheet #1 introduces you to what Player-Missile graphics is really all about.

Player-Missile Worksheet #1

Most people are unaware of the true nature of players and missiles. Let's start by finding out what they really look like. Insert the Advanced Topics Diskette and type:

```
RUN "D:INTRO.PM"     (READY will appear on your screen.)
```

Don't concern yourself for now about how this program works. You can now move some players and missiles onto the screen. Type the following in direct mode. (Type a zero after the "P", not the letter "o".)

```
POKE P0,60
POKE P1,90      These are the four players.
POKE P2,140
POKE P3,180

POKE M0,100
POKE M1,120     These are the four missiles.
POKE M2,80
POKE M3,160
```

As you can see, players and missiles are nothing more than strips from top to bottom on the screen. Players are simply four times as wide as missiles. Try moving the players and missiles around a little by POKEing numbers into P0 through P3 and M0 through M3. Try something like this:

```
FOR I=0 TO 200:POKE P0,I:NEXT I
```

Or try this:

```
FOR I=0 TO 200:POKE P0,I:POKE M0,200-I:NEXT I
```

When you finish experimenting, make sure all of your players and missiles are visible on the screen.

Now let's try something different. Type the following:

```
FOR I=768 TO 2047:POKE PM*256+I,0:NEXT I
```

The players and missiles are still there, but now they are "blanked out". Later we'll explore what's happenning here. For now, try doing the following (you can use the arrow keys to make changes in the statement above without retyping it each time):

```
FOR I=768 TO 2047:POKE PM*256+I,170:NEXT I
```

Experiment with other numbers in place of 170.

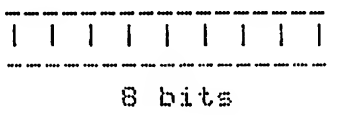
Player-Missile RAM

An important concept that you must understand to use Player-Missile graphics is how players and missiles are stored in computer RAM. In this section the relationship between players and missiles and computer memory are explained.

Before exploring Player-Missile RAM, you must have a complete understanding of bytes and bits, and the binary (base 2) number system.

If your computer has 48K RAM, that means there are 48K (which equals 48 x 1024) cells of memory. A byte is simply a number which each of these memory cells can hold. These numbers can only be in the range 0 to 255. See the advanced module on "Internal Representation of Text and Graphics" if you are interested in exploring how the computer can do all the things you see it do with only the capability of storing numbers from 0 to 255.

Diagram 1



The bytes placed in computer memory are made up of 8 smaller units called bits (binary digits). Each bit is either a zero or a one -- zero and one are the two digits used in the binary number system.

In base ten there are 10 digits used: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The number 746 in base ten ha a 6 in the ones place (6 x 1) and a 4 in the tens place (4 x 10) and a 7 in the hundreds place (7 x 100). In base ten each position has a place value. Base two (binary number system) follows the same principles (see Diagram 2).

Diagram 2

Base 10

| 6 | 3 | 0 | 9 | <--- Digits 0 - 9

place values 10^3 10^2 10^1 10^0

1000 100 10 1

Base 2

| 1 | 0 | 1 | 1 | <--- Digits 0 - 1

place values 2^3 2^2 2^1 2^0

8 4 2 1

Try filling in the blanks in the following problem which converts a binary number into its decimal equivalent.

Binary number	1	0	0	1	0	1	1	0
place value	128	___	___	___	8	4	2	1

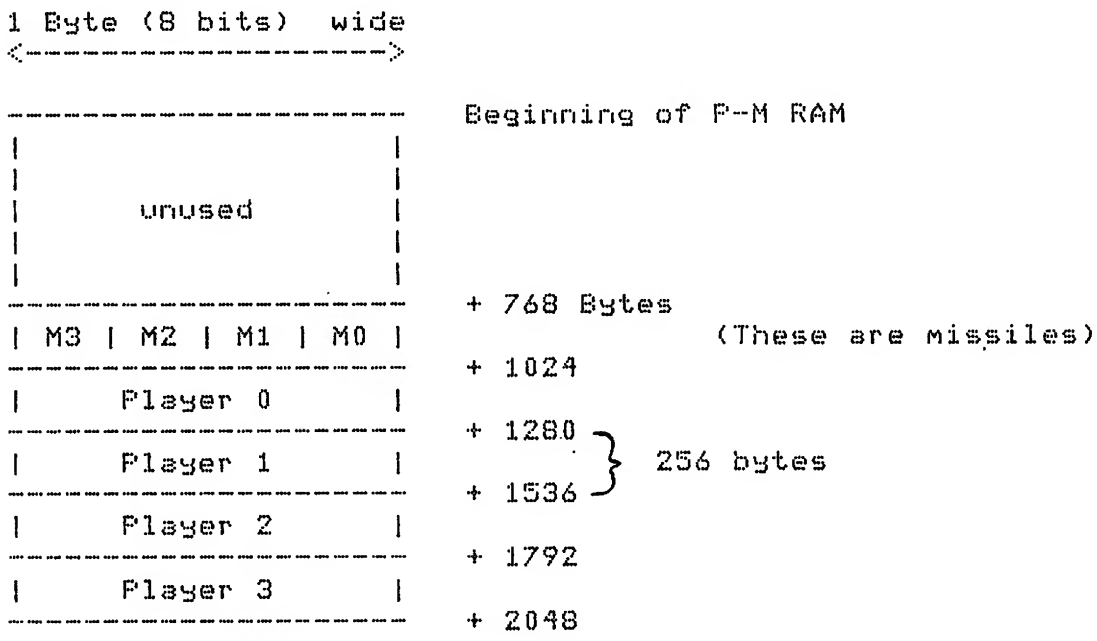
Decimal equivalent = 1 × 128 +
0 × ___ +
0 × ___ +
__ × ___ +
__ × 8 +
__ × ___ +
__ × ___ +
__ × ___
= 150

Like all other memory locations, Player-Missile RAM is just a sequence of bytes used for a special purpose. Each byte is a value from 0 to 255. The reason these memory locations are called Player-Missile RAM is because the shapes and vertical location of each player and each missile are determined by the numbers that are placed in this portion of memory.

Player-Missile RAM is not restricted to a particular location in memory, rather it can be almost anywhere you want it to be. The computer, however, always needs a way to find where this RAM is so that it can look there to decide what shape players and/or missiles to put on the screen.

Diagram 3 shows how Player-Missile RAM is organized for single resolution players (more about resolution later). 2048 bytes of RAM (2K) are needed.

Diagram 3



Note that each player is 1 byte (8 bits) wide while each missile is only 2 bits wide. This is why missiles are four times thinner than players. Note also that each player uses 256 bytes of RAM. Worksheet #2 explores how these 256 bytes are used to make a player on the screen.

Player-missile Worksheet #2

Start by inserting the Advanced Topics Diskette and once again running "D:INTRO.PM". Then do the following steps:

```
PM=256*PM
```

The variable PM now holds the address of the beginning of our Player-Missile RAM.

```
POKE P1,100
```

This moves Player 1 onto the screen. Let's clear out all of Player 1's RAM which begins 1280 bytes after the beginning of P-M RAM (see Diagram 3).

```
FOR I=PM+1280 TO PM+1535:POKE I,0:NEXT I
```

Now let's try POKEing some different numbers into Player 1's RAM.

	Decimal	Binary
POKE PM+1320,24	24	00011000
POKE PM+1321,60	60	00111100
POKE PM+1322,60	60	00111100
POKE PM+1323,153	153	10011001
POKE PM+1324,255	255	11111111
POKE PM+1325,24	24	00011000
POKE PM+1326,24	24	00011000
POKE PM+1327,24	24	00011000
POKE PM+1328,102	102	01100110

```
FOR I=0 TO 200:POKE P1,I:NEXT I
```

Each player has a horizontal position determined by a number POKEd into a special location. The variable P1 contains the location used to determine Player 1's horizontal position. (If you PRINT P1, you see Player 1's horizontal position register is location 53249 -- see chart in back.) Experiment with POKEing different numbers into his location to see where the player ends up on the screen (numbers less than about 42 put the player off the left edge).

Try the following:

```
5 FOR I=46 TO 200:POKE P1,I:NEXT I:FOR I=200 TO 46 STEP
-1:POKE P1,I:NEXT I:GOTO 5
```

GOTO 5 (No line number)

Then try to create your own shape and POKE the numbers into Player-Missile RAM.

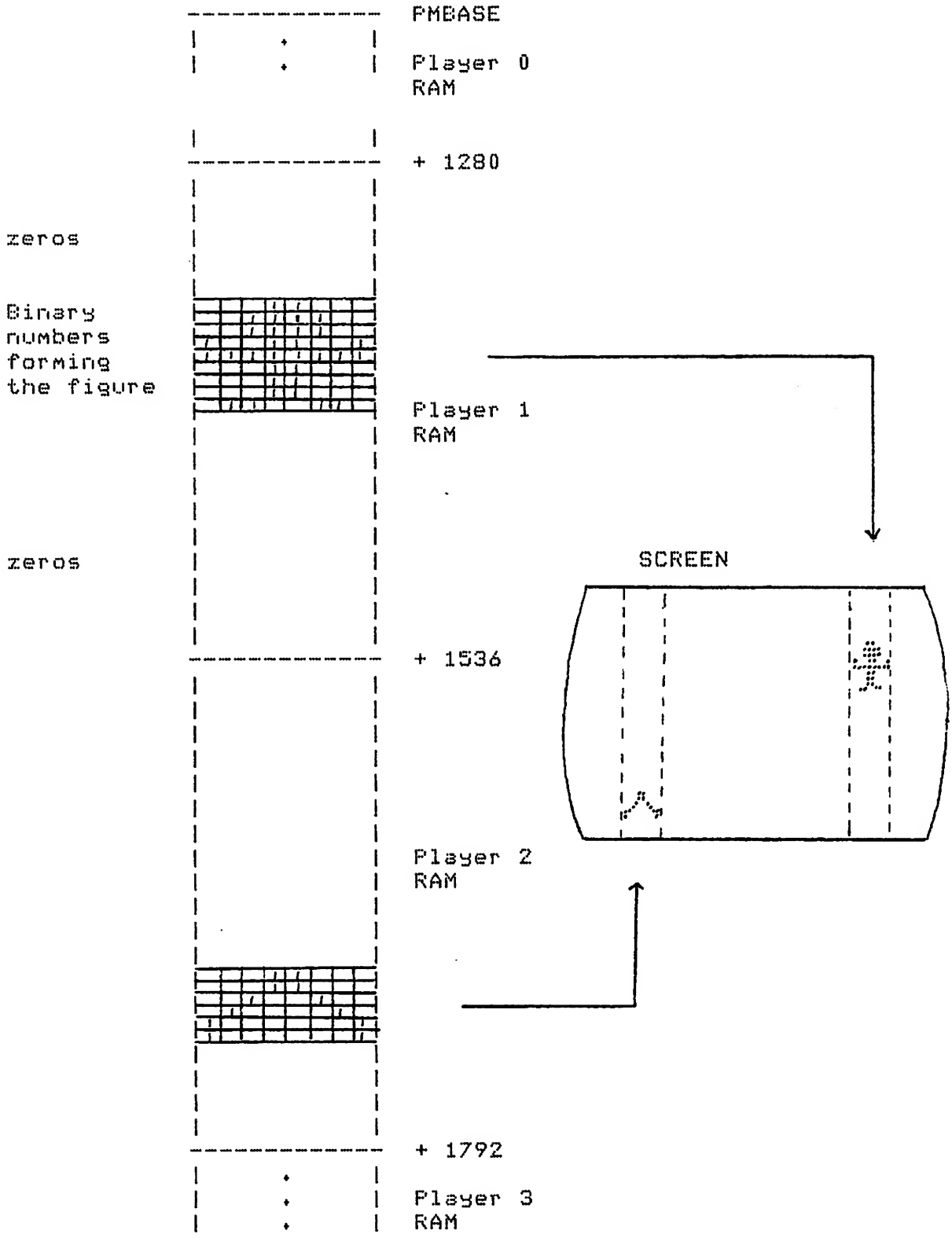
Vertical Motion of Players

Moving an object formed out of a player vertically up and down the screen is relatively difficult and can be quite slow. Assembly language is needed to obtain smooth motion up and down, but in this section we'll explore how to accomplish reasonable vertical motion using only BASIC.

In worksheet #2 you discovered that you can make an object on the screen by putting zeros into most of a player's RAM (blanking it out) and then turning on certain bits by POKEing numbers into certain locations in the player's RAM (see Diagram 4).

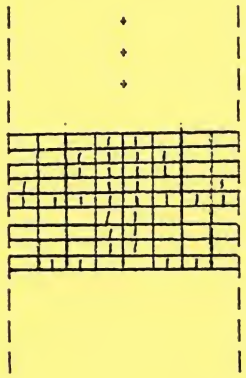
If the bits that are turned on (the ones) are near the bottom of the player's RAM, then the figure will show up near the bottom of the screen. Similarly, if the non-zero bytes are placed closer to the top of the player's RAM then the figure will be higher up on the screen. To move the figure up or down on the screen, one simply has to move the non-zero bytes up or down in the player's RAM. Worksheet #3 shows you how to do this.

Diagram 4



Player-Missile Worksheet #3

In worksheet #2 you created a little man on the screen and moved him horizontally. In this worksheet you'll bring the man back on the screen and learn to move him vertically. To get the man (Player 1) back on the screen, insert the Advanced Topics Diskette and type: RUN "D:MAN.PM".



PM+1320 The man is stored in memory (Player 1 RAM) beginning at location PM+1320 and ending at location PM+1328.

PM+1328

Let's write a subroutine which copies all of the bytes down one notch in memory. The variable START will hold the location in memory that the man starts at (location PM+1320 in this example). Our man is stored in the bytes from START to START+8. To move him down, it is necesssary to copy these bytes to locations START+1 to START+9. The following subroutine does this.

```
500 REM MOVE DOWN
510 FOR BYTE=START+8 TO START STEP -1
520 POKE BYTE+1,PEEK(BYTE)
530 NEXT BYTE
540 REM Erase the starting byte so as not to leave a trail.
550 POKE START,0
560 REM Change START to the new starting byte.
570 START=START+1
580 RETURN
```

Now all that needs to be done is for START to be set to the correct beginning value and then we can use the subroutine. Try the following:

```
START=PM+1320
FOR I=1 TO 100:GOSUB 500:NEXT I
```

Before continuing, write a subroutine starting at line number 600 that moves the man up one byte. It will resemble the subroutine that moves him down, but it will copy bytes starting at the top of the player rather than the bottom. Be careful -- there are some other minor differences!

See if you can write the necessary BASIC to enable you to move the man around the screen with a joystick. Remember that the variable P1 is the location to POKE to change the man's horizontal position. Unfortunately, you can not PEEK in this location so you have to be a little tricky to keep track of the man's horizontal position (Hint: use another variable just for this purpose). You can run an example of this program stored in "D:MOVEMAN.PM".

Setting Up Player-Missile Graphics

In this section you will learn how to turn on the Player-Missile Graphics feature of the Atari. You will also learn how to create different resolution players and how to change their width and color.

You have seen that Player-Missile graphics requires that you set aside 2K of RAM which is called Player-Missile RAM (see Diagram 3). It is common to reserve this RAM at the end of memory by making the computer think that it has less memory than it really does. That enables you to use the extra memory for your own uses.

Memory location 106 tells the computer how many pages of RAM it has. (1 page = 256 bytes. 4 pages = 1K.) Thus, if you subtract 8 pages from the total indicated in location 106, you will reserve 2K of RAM for Player-Missile graphics. The following accomplishes this:

```
PM=PEEK(106)-8:POKE 106,PM:GRAPHICS 0
```

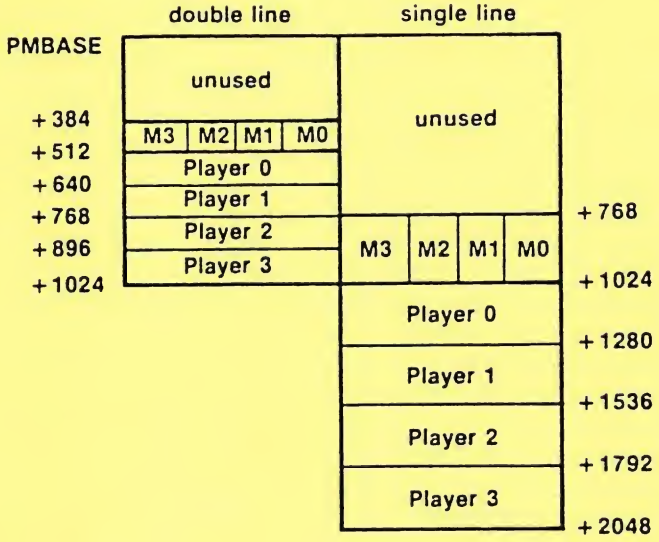
Note: The GRAPHICS 0 statement is necessary so that the computer moves the location of some things that it had already placed in the 8 pages that you are stealing -- in particular, screen RAM and the display list.

Continue now by completing worksheet #4.

Player-Missile Worksheet #4

Players and missiles can be either single or double resolution. Double resolution players use up half as much memory (1K) as single resolution players, but their design is less precise as you will see below. Diagram 5 shows you the layout of Player-Missile RAM for both resolutions.

Diagram 5



Type in the following program in order to explore the difference between single and double resolution players.

```
10 PMBASE=54279
```

(This is the computer location where the beginning address of P-M RAM must be stored.)

```
20 PM=PEEK(106)-8:POKE 106,PM:GRAPHICS 0
30 POKE PMBASE,PM
```

(POKE in the address -- actually the page number -- of where Player-Missile RAM will begin.)

```
40 PM=256*PM
```

(Store the actual address, rather than the number of pages, in PM for later use.)

```
50 FOR I=0 TO 2047:POKE PM+I,0:NEXT I
```

(Clear out all of P-M RAM.)

```
60 P1=53249:POKE P1,150
```

(This is the horizontal position register of Player 1. POKEing a number in this location changes the horizontal position of the player.)

```
70 POKE 53277,3
```

(This turns on Player-Missile graphics. See also lines 180 and 190.)

```
80 FOR I=0 TO 8
90 READ BYTE:POKE PM+1400+I,BYTE
100 NEXT I
110 DATA 24,60,60,153,255,24,24,24,102
```

(This puts player data into Player 1's single resolution RAM. The following puts the same data into the RAM area for double resolution players. See Diagram 5.)

```
120 RESTORE
130 FOR I=0 TO 8
140 READ BYTE:POKE PM+700+I,BYTE
150 NEXT I
```

(Finally, let's ask which resolution player the user wants to see.)

```
160 PRINT "Resolution 1 or 2";
170 INPUT RES
180 IF RES=1 THEN POKE 559,62
```

(This POKE turns on single resolution P-M graphics.)

```
190 IF RES=2 THEN POKE 559,46
```

(This POKE turns on double resolution P-M graphics.)

```
200 GOTO 160
```

Now try running the program. Try writing some code that moves the player around the screen in both resolutions. It's easy to move it horizontally in either resolution -- just POKE a new number into P1. But moving it vertically in both

resolutions simultaneously requires that you shift bytes in two places in memory -- single resolution player RAM and double resolution player RAM.

Let's experiment with some other properties of players. First type: RUN "D:PLAYER.PM". Then try the following POKES.

```
POKE 559,46          (Changes to double resolution.)
POKE 559,62          (Changes back to single resolution.)
WIDTH=53257          (Location determining Player 1's width.)
POKE WIDTH,1         (Double size player.)
POKE WIDTH,3         (Quadruple size player.)
POKE WIDTH,0         (Back to normal size.)

POKE 559,46
POKE WIDTH,1

COL=705              (Location of Player 1's color.)
POKE COL,14
```

Try POKEing other numbers between 0 and 255 into COL. Play with changing the resolution, width, color, and position of your player.

Note: Much of the work involved in designing and moving players can be avoided by using one of the many software utility porgrams available for creating players. Experiment with some of these Player-Missile Graphics development tools (see PLAYER MAKER by Wayne Harvey).

Summary and Challenges

The Atari computer has four color registers, besides for the background color, that are used to draw graphics in the different graphics modes available. These color registers are sometimes referred to as playfields and are labelled Playfield 0 (PF0), PF1, PF2, and PF3. In GRAPHICS 0 and 8 it turns out that most of the screen (all of the blue) comes from PF2.

When you use players, sometimes you want them to move on top of other things you have drawn on the screen and sometimes you want them to move behind the playfields. You do have some control over these priorities.

If you always want all of the players to be on top of everything else, then POKE location 623 with a 1. If you always want the players beneath other things drawn on the screen then you will want to POKE 623,4 (see chart below). Note that if you put the players beneath the playfields, then in GRAPHICS 0 and 8 you wouldn't even see them because the entire screen is Playfield 2.

Priorities

P0-P3, PF0-PF3, background	POKE 623,1
P0-P1, PF0-PF3, P2-P3, background	POKE 623,2
PF0-PF3, P0-P3, background	POKE 623,4
PF0-PF1, P0-P3, PF2-PF3, background	POKE 623,8

You also have the capability of detecting if a player or missile collided with some other player or with a playfield (meaning something drawn on the screen with some color register). The chart below shows the locations to PEEK in. If the value read is a 0, 1, 2, or 4, then the collision occurred with player or playfield 0, 1, 2, or 3. (Examples follow the chart.)

53248	Missile 0 to playfield collision
53249	Missile 1 to playfield collision
53250	Missile 2 to playfield collision
53251	Missile 3 to playfield collision
53252	Player 0 to playfield collision
53253	Player 1 to playfield collision
53254	Player 2 to playfield collision
53255	Player 3 to playfield collision

```
53256     Missile 0 to player collision
53257     Missile 1 to player collision
53258     Missile 2 to player collision
53259     Missile 3 to player collision

53260     Player 0 to player collision
53261     Player 1 to player collision
53262     Player 2 to player collision
53263     Player 3 to player collision
```

Examples:

```
IF PEEK(53253)=2 THEN  Player 1 collided with Playfield 2.
IF PEEK(53258)=0 THEN  Missile 2 collided with Player 0.
IF PEEK(53262)=4 THEN Player 2 collided with Player 3.
```

After a collision occurs, you can only clear the register by doing a: POKE 53278,1.

To set the horizontal position of players or missiles, POKE the appropriate register with a number from 0 to 255. Only numbers from about 44 to 200 will put players or missiles on the visible screen. Thus, by POKEing a zero into the horizontal position register, you can get rid of players or missiles from the screen. That's the easiest way to "clear" them off the screen.

```
53248     Horizontal position of Player 0
53249     Horizontal position of Player 1
53250     Horizontal position of Player 2
53251     Horizontal position of Player 3

53252     Horizontal position of Missile 0
53253     Horizontal position of Missile 1
53254     Horizontal position of Missile 2
53255     Horizontal position of Missile 3
```

The following locations can be used to set the width of players. POKEing a zero into these registers gives normal width, a 1 is double width, and a 3 is quadruple width.

```
53256     Width of Player 0
53257     Width of Player 1
53258     Width of Player 2
53259     Width of Player 3
```

The following locations are used to change colors.

```
704       Color of Player/Missile 0
705       Color of Player/Missile 1
706       Color of Player/Missile 2
707       Color of Player/Missile 3
```

Finally, the following code is used to setup Player-Missile Graphics.

```
100 PM=PEEK(106)-8:POKE 106,PM:GRAPHICS 0
110 POKE 54279,PM:PM=256*PM
120 POKE 53277,3:POKE 559,62
```

(Note: On line 120: POKE 559,46 for double resolution.)